Practical, Distributed Network Coordinates

Russ Cox, Frank Dabek, Frans Kaashoek, Jinyang Li, Robert Morris
MIT Laboratory for Computer Science
$\{$*rsc, fdabek, kaashoek, jinyang, rtm*$\}$*@lcs.mit.edu*

ABSTRACT – *Vivaldi is a distributed algorithm that assigns synthetic coordinates to Internet hosts, so that the Euclidean distance between two hosts' coordinates predicts the network latency between them. Each node in Vivaldi computes its coordinates by simulating its position in a network of physical springs. Vivaldi is both distributed and efficient: no fixed infrastructure need be deployed and a new host can compute useful coordinates after collecting latency information from only a few other hosts. Vivaldi can rely on piggy-backing latency information on application traffic instead of generating extra traffic by sending its own probe packets.*

This paper evaluates Vivaldi through simulations of 750 hosts, with a matrix of inter-host latencies derived from measurements between 750 real Internet hosts. Vivaldi finds synthetic coordinates that predict the measured latencies with a median relative error of 14 percent. The simulations show that a new host joining an existing Vivaldi system requires fewer than 10 probes to achieve this accuracy. Vivaldi is currently used by the Chord distributed hash table to perform proximity routing, replica selection, and retransmission timer estimation.

1 Introduction

Synthetic coordinate systems are an approach to predicting inter-host Internet latencies. Nodes compute synthetic coordinates such that the Euclidean distance between the synthetic coordinates of different nodes predict latency in the Internet. Thus, if a node x learns about the coordinates of a node y with which it hasn't communicated before, x doesn't have to perform an explicit measurement to determine the latency to y; instead, the Euclidean distance between x and y in the space is an accurate predictor of the latency.

The ability to predict latency without prior communication allows systems to use proximity for better performance with less measurement overhead. A coordinate system could be used to select which of a number of replicated servers to fetch a data item from; such a system is particularly helpful when the number of potential servers is large or the amount of data is small. In either case it would not be practical to first probe all the servers to find the closest, since the cost of the probes would outweigh the benefit of an intelligent choice. Content distribution and file-sharing systems such as KaZaA [10], BitTorrent [2], and CoDeeN [23] are examples of systems that offer a large number of replica servers. CFS [5] and DNS [12] are examples of systems that offer modest numbers of replicas, but each piece of data is small.

GNP demonstrated that it is possible to calculate synthetic coordinates, and that they can be used to predict Internet latencies [14]. GNP relies on a small number (5-20) of "landmark" nodes; other nodes measure latency to the landmarks to help them choose coordinates. The choice of which nodes are used as landmarks can significantly affect the accuracy of latency predictions made by GNP.

Vivaldi is a simple, distributed, symmetric algorithm for computing synthetic coordinates that requires no landmarks and provides an accuracy similar to that of GNP. In Vivaldi, each node computes coordinates for itself. Each time a node communicates with another node, it measures the latency to that node, and then adjusts its coordinates to minimize the error between measured latencies and predicted latencies. Vivaldi requires the user to set only a single parameter, which describes how much to adjust a node's coordinates in response to one new latency sample. This parameter is largely independent of the input and can be set conservatively to ensure accuracy at the cost of convergence time.

We believe that Vivaldi's properties could make synthetic coordinates more widely applicable. Vivaldi's simplicity, accuracy, and distributed, symmetric nature align well with the requirements of peer-to-peer systems that do not inherently have special, reliable nodes that are candidates for landmarks. As an example of its applicability, we describe how Vivaldi can be used to perform proximity routing and server selection, and to set retransmission timers in the Chord DHT [21].

2 Design

Vivaldi assigns each node synthetic coordinates in a D-dimensional space. The goal of Vivaldi is to assign coordinates so that the Euclidean distance in synthetic coordinate space between two hosts accurately predicts the round-trip latency of packet transmission between the hosts.

Vivaldi chooses coordinates by sampling the network latency between each node and a few other nodes, and adjusting the nodes' coordinates to minimize the error between the predicted and sampled latencies. We first describe a centralized algorithm to minimize the error and then generalize it to a practical, distributed algorithm.

2.1 Centralized algorithm

When formulated as a centralized algorithm, the input to Vivaldi is a matrix of real network latencies M, such that M_{xy}

This research was conducted as part of the IRIS project (`http://project-iris.net/`), supported by the National Science Foundation under Cooperative Agreement No. ANI-0225660. Russ Cox is supported by a fellowship from the Fannie and John Hertz Foundation.

is the latency between x and y. The output is a set of coordinates. Finding the best coordinates is equivalent to minimizing the error (E) between predicted distances and the supplied distances. We use a simple squared error function:

$$E = \sum_x \sum_y (M_{xy} - dist(x, y))^2$$

where $dist(x, y)$ is the standard Euclidean distance between coordinates of x and y.

Vivaldi uses an algorithm based on a simulation of a network of physical springs to minimize E. This algorithm was inspired by work on model reconstruction [9]; it mirrors a similar recent approach using force fields [20]. Conceptually, Vivaldi places a spring between each pair of nodes for which it knows the network latency, with the rest length set to that latency. The length of each spring is the distance between the current coordinates of the two nodes. The potential energy of a spring is proportional to the displacement from its rest length squared: this displacement is identical to the prediction error of the coordinates. Therefore minimizing the potential energy of the spring system corresponds to minimizing the prediction error E.

Simulating spring relaxation requires much less computation than more general optimization optimization algorithms such as the simplex algorithm (used by GNP) and produces similarly accurate results. The spring-based algorithm outperforms simplex mainly because it takes advantage of gradient information to move the solution toward a minimal error solution; simplex does not depend on such gradient information and explores the the solution space in a less directed manner.

Vivaldi simulates the physical spring system by running the system through a series of small time steps. At each time step, the force on each node is calculated and the node moves in the direction of that force. The node moves a distance proportional to the applied force and the size of the time step. Each time a node moves it decreases the energy of the system; however, the energy of the system stored in the springs will typically never reach zero since network latencies don't actually reflect an Euclidean space. Neither the spring relaxation nor the simplex algorithm is guaranteed to find the global minimal solution; both can converge to a local minimum.

2.2 Distributed calculation

In the distributed version of Vivaldi, each node simulates a piece of the overall spring system. A node maintains an estimate of its own current coordinates, starting at the origin. Whenever two nodes communicate, the two nodes measure the latency between them and exchange their current synthetic coordinates. In RPC-based systems, this measurement can be accomplished by timing the RPC; in a stream oriented system, the receiver might echo a timestamp. An application might choose to make several measurements and report the minimum (or median) to Vivaldi, however, in the current deployment of Vivaldi on the Chord distributed lookup system (see Section 4) we report the latency of each RPC to Vivaldi without degrading performance.

```
// called for each new measurement.
// s_c is the other host's coordinates.
// s_l is the one-way latency to that host.
// δ starts at 1.0.
update(s_c, s_l) {
  // unit vector towards other host
  Vector dir = s_c - my_c;
  dir = dir / length(dir);
  //Distance from spring's rest position
  d = dist(s_c, my_c) - s_l;
  // displacement from rest position
  Vector x = dir * d;
  // reduce δ at each sample
  δ -= 0.025;
  // but stop at 0.05
  δ = max (0.05, δ);
  x = x * δ;
  // apply the force
  my_c = my_c + x;
}
```

Figure 1: Pseudo-code for the Vivaldi update routine. `update()` moves the node's coordinates (`my_c`) based on the measured latency to another node and the other node's current synthetic coordinates.

Once a measurement is obtained, both nodes adjust their coordinates to reduce the mismatch between the measured latency and the coordinate distance (see Figure 1). A node moves its coordinates towards a point p along the line between it and the other node. The point p is chosen to be the point which reduces the difference between the predicted and measured latency between the two nodes to zero. To avoid oscillation a node moves its coordinates only a fraction δ towards p.

A node initializes δ to 1.0 when it starts, and reduces it each time it updates its coordinates. Vivaldi starts with a large δ to allow a node to move quickly towards good coordinates, and ends up with a small δ to avoid oscillation.

Figure 1 omits one detail: if two nodes have the same coordinates (the origin, for instance), they each choose a random direction in which to move.

We expect that applications using Vivaldi will contact other nodes in the ordinary course of events, and report latency information to Vivaldi after each such contact. This means that Vivaldi will not need to send any packets itself. It also means that applications must add space for Vivaldi coordinates to their packet formats. The application should ensure that all nodes sample at roughly the same rate.

3 Evaluation

This section uses simulations to explore Vivaldi's performance, focusing on how quickly Vivaldi converges and on how well Vivaldi's coordinates predict Internet latency.

Unless otherwise noted, we perform the simulations as follows. Each simulation involves 750 nodes. Each node starts with its synthetic location at the origin. Nodes take latency samples from randomly chosen other nodes. Synthetic coordinates have 5 dimensions; more dimensions provide better accuracy, but the improvement is small after two dimensions. This result is supported by principle component analysis on the matrix of latencies (omitted here but available separately [4]), and by similar observations by Ng and Zhang [14].

3.1 Latency data

The Vivaldi simulations are driven by a matrix of inter-host Internet latencies; Vivaldi uses a subset of the latencies for its samples, and the full matrix is needed to evaluate the quality of Vivaldi's resulting predictions. Since deriving realistic latencies from Internet topology models is difficult, we chose to use measured latencies.

We built a tool based on the King method [8] to collect a full matrix of latencies among 750 Internet DNS servers. To determine the distance between DNS server A and server B we first measure the round trip time to server A and then ask server A to recursively resolve a domain served by B. The difference in times between the two operations yields an estimate of the round trip time between A and B. We use half the round trip time as the latency.

We harvested the addresses of recursive DNS servers by extracting the NS records for IP addresses of hosts participating in a Gnutella network. If a domain is served by multiple, geographically diverse name servers, queries targeted at domain D (and intended for name server B) could be forwarded to a different name server, C, which also serves D. Our tool cannot control where queries are forwarded. To avoid this error, the list of target domains and name servers was filtered to include only those domains where all authoritative name servers are on the same subnet (i.e. the IP addresses of the name servers are identical except for the low octet).

An asynchronous tool was used to determine the $N(N-1)$ pair-wise latencies using the King method. The final latency for a given pair was taken to be the minimum of 10 trials, in order to filter out queuing delays and misses in DNS server caches. Collecting all pairwise latencies required several hours. Figure 2 shows the cumulative distribution of latencies produced from this data set. Because they are name servers, the machines included in the King trace are likely to be well connected to the Internet. The servers are geographically diverse, however, and include machines in North America, Europe, and Asia. For comparison, Figure 2 also shows pair-wise latencies obtained by direct measurements of 192 PlanetLab hosts [1]. The King data has higher median latency (100 msec) than the PlanetLab data set (75 msec); this is likely due to the fact that most PlanetLab hosts are located at North American universities with fast Internet2 connections.

We used the $N(N-1)$ latencies produced by the King tool as the input to a packet-level peer-to-peer simulator [7].

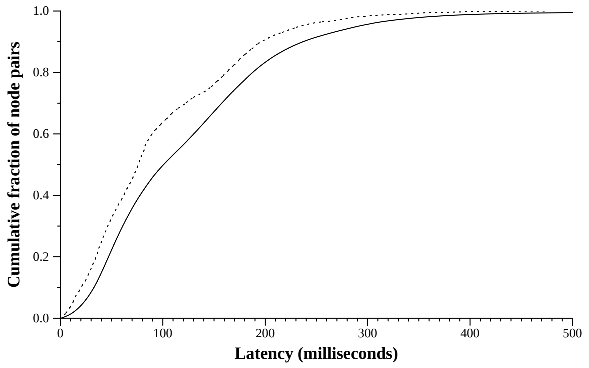

Figure 2: The cumulative distribution of pairwise round-trip latencies in the King data set (solid line) and the PlanetLab data set (dotted line)

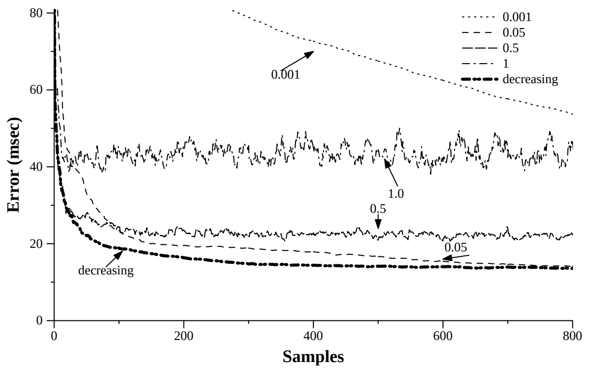

Figure 3: A comparison of Vivaldi's convergence time using different, fixed values of δ. Curves plot the error after the given number of samples; the error is calculated as the median of all round-trip pair-wise differences between predicted and actual latency. The lines marked with numbers indicate performance with the given fixed δ; the bold line marked "decreasing" shows the actual algorithm's performance.

The simulator delays each RPC packet by the time specified in the King data. Each node runs an instance of Vivaldi which sends RPCs to other nodes, measures the RPCs' RTTs, and uses those RTTs to update its synthetic coordinates.

3.2 Setting the timestep

The δ variable (the timestep used during the spring simulation) in Figure 1 affects how fast Vivaldi converges. Figure 3 compares Vivaldi's performance with a range of fixed δ values against the actual algorithm's slowly decreasing δ. Each curve in Figure 3 shows results from a simulation with a different fixed δ. Each simulation begins with all 750 nodes starting Vivaldi at the same time. The x-axis reflects how much time has passed; the units are the number of samples each node has taken. The y-axis shows median error over all pair-wise predictions.

Figure 3 shows that small δ values (such as 0.001) result in long convergence times. Intermediate values (as large as

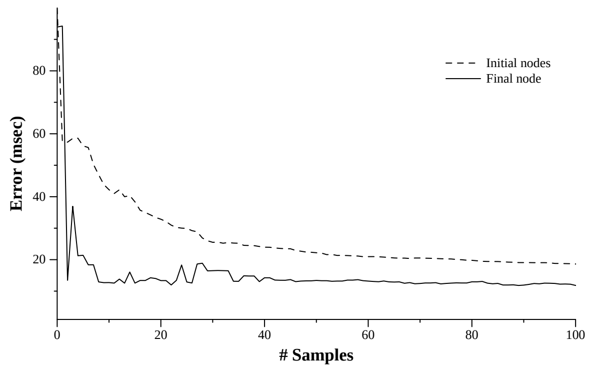

Figure 4: This figure compares Vivaldi's convergence time when all nodes join at once with the convergence time when a few nodes join an existing converged system. The solid line shows a node joining an already stable system: the node converges after collecting fewer than 10 samples.

0.5) result in much faster convergence at the cost of some accuracy. As δ increases, final accuracy decreases, since large δ values allow nodes to vibrate more around their "correct" positions. Very large δ values (such as 1.0) cause the system to oscillate and fail to converge.

The bold line marked "decreasing" in Figure 3 shows the performance with a decreasing δ. The initial value is large, so the error decreases quickly. δ eventually becomes small, so the error converges to a low value and does not oscillate. Using the decreasing δ (with a minimum of 0.05), the median error drops below 20 ms after 70 samples. When δ is fixed at 0.05 from the start of the simulation, the error does not drop below 20 ms until sample 157.

Vivaldi has no user-tunable parameters other than δ. This simplicity makes the algorithm more robust and easier to deploy.

3.3 Time to Convergence

Figure 3 also shows that when all nodes join at the same time with incorrect initial coordinates, the system converges slowly (the median error is still dropping after 500 timesteps). Many applications of Vivaldi are likely to involve new nodes joining a larger existing system; thus new nodes are likely to join a system whose Vivaldi coordinates have already converged.

Figure 4 shows what happens when 10 new nodes join a converged Vivaldi system of 740 nodes. The solid line shows the error for one of the 10 nodes. The dashed line shows how long 740 nodes take to converge when they all start at once. New nodes arrive at good coordinates with less than ten samples. The algorithm's large initial δ allows the new node to converge rapidly in this case. The spike in the solid line around sample three is caused by the fact that the initial δ is probably too large and allows a brief initial period of oscillation.

3.4 Vivaldi Accuracy

To understand the accuracy of Vivaldi we measured the error in the prediction of each pairwise latency using a relative error metric [14]:

$$\frac{\text{abs}(\text{measured} - \text{predicted})}{\text{min}(\text{measured}, \text{predicted})}$$

Figure 5 shows the distribution of relative errors between every pair of nodes for Vivaldi. The error (solid line) is measured after the system has converged. In this experiment, Vivaldi's communication pattern was not restricted; the results determine the best-case performance of Vivaldi. Also shown is the error when Vivaldi is limited to communication with 64 neighbors (dot-dash line) and 32 neighbors (dashed line). Reducing the number of neighbors a node communicates with reduces the accuracy of predictions. The median error in the unrestricted case is 14 percent; using 64 neighbors the error is 22 percent; using 32 neighbors the error is 30 percent. Determining the exact influence of communication patterns on Vivaldi's performance is an area of future work.

3.5 Comparison with GNP

To understand Vivaldi's performance, we perform an initial comparison to GNP by running the GNP software [13] on our data sets. GNP's performance depends on the placement of landmarks in the network. To measure the effect of landmark placement, we performed 30 experiments, each with a random set of 32 landmarks. For each experiment we produced a cumulative distribution of errors; the dotted line shows the median of the values of those distributions at a given cumulative fraction. Error bars denote the highest and lowest values and a given fraction.

The variance in GNP prediction error in Figure 5 shows that GNP's performance depends on the appropriate placement of landmarks. It is not clear in practice how to place landmarks appropriately. The authors of GNP suggest a clustering technique that depends on *a priori* knowledge of the complete latency matrix, but the running time of the provided implementation of this algorithm prevented us from running it on our data sets. One of Vivaldi's advantages is symmetry: no nodes need be selected as landmarks.

Vivaldi's best performance (when nodes' communication is not restricted) is better than GNP's best performance. To better compare the two systems, we examine the error when nodes running Vivaldi communicate with 32 random neighbors (GNP was run with 32 landmarks). In this test, Vivaldi's median error (30 percent) closely matches the best median error of GNP (28 percent).

This comparison illustrates that Vivaldi's prediction error is competitive with that of GNP. A more detailed comparison of the two systems is future work.

4 Applications

We believe Vivaldi will be useful for a variety of distributed applications such as CDNs, the DNS, and file-sharing appli-

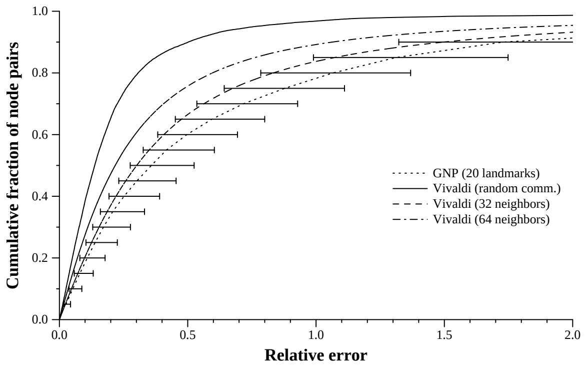

Figure 5: Vivaldi and GNP accuracy. The solid line shows Vivaldi's accuracy when nodes' communication is unrestricted. The dashed line shows Vivaldi's accuracy when nodes are restricted to communicating with 32 neighbors. Several GNP experiments were run, each with a different set of 32 random landmarks. The dotted line denotes the median result, the error bars show the best and worst prediction errors.

cations such as KaZaA. We developed Vivaldi to assist proximity routing and server selection in the Chord [21] peer-to-peer lookup system. We will describe that application in detail here.

Chord uses coordinates to efficiently build routing tables based on proximity so that lookups are likely to proceed to nearby nodes. A node receives a list of candidate nodes and selects the one that is closest in coordinate space as its routing table entry; coordinates allow the node to make this decision without probing each candidate.

Chord utilizes coordinates when performing an iterative lookup. When a node n_1 initiates a lookup and routes it through some node n_2, n_2 chooses a next hop that is close to n_1 based on Vivaldi coordinates. In an iterative lookup n_1 sends an RPC to each intermediate node in the route, so proximity to n_1 is more important than proximity to n_2.

A new version of DHash [5], a distributed hash table built on top of Chord, uses Vivaldi to perform server selection. DHash uses erasure coding to divide blocks into a number of fragments. Only a subset of these fragments are necessary to reconstruct the original block. A node fetches a block by asking the block's successor for a list of the nodes holding the fragments, along with the coordinates of those nodes. The fetching node then fetches fragments from the nodes with the closest coordinates. If the node had to first measure the latency to each of these nodes, the extra round-trip time would probably cancel out the benefit of choosing the closest fragments.

Chord (and DHash) also use Vivaldi in their RPC system. Chord sends RPCs over UDP, so Chord must handle retransmissions. Chord often contacts other nodes just once, so it cannot profitably measure the round-trip time in order to set the RPC retransmission timer. Instead, Chord uses a small multiple of the latency predicted by Vivaldi as the initial RPC retransmission timer.

Chord and DHash required a few modifications to use Vivaldi. Whenever one node sends an RPC request or reply to another, it includes its own coordinates. The RPC system times each RPC and tells Vivaldi the measured latency and the coordinates of the other node. This allows Vivaldi to collect information without much added overhead, since the coordinates increase the size of each RPC message by just 12 bytes. In addition, whenever nodes exchange routing information about other nodes, they send along the coordinates of those other nodes as well as their IP addresses. Thus Chord always knows the coordinates of any node it is about to contact, even if it has never talked to that node before.

After we modified DHash to perform proximity routing and replica selection using Vivaldi the time required to fetch a block decreased by a factor of two when the system was run on the PlanetLab test bed.

5 Related work

Vivaldi was inspired by GNP [14], which demonstrated that coordinates can effectively describe the latencies between hosts on the Internet. Vivaldi's contribution is a distributed algorithm that computes coordinates without landmark nodes.

IDMaps [6] is a proposed infrastructure to help hosts predict Internet latency to each other. The IDMaps infrastructure consists of a few hundred or thousand *tracer* nodes. Every tracer measures the Internet latency to every other tracer. The tracers also measure the latency to every CIDR address prefix, and jointly determine which tracer is closest to each prefix. Then the latency between host h_1 and host h_2 can be estimated as the latency from the prefix of h_1 to that prefix's tracer, plus the latency from the prefix of h_2 to that prefix's tracer, plus the latency between the two tracers. One advantage of IDMaps over Vivaldi is that IDMaps reasons about IP address prefixes, so it can make predictions about hosts that are not even aware of the IDMaps system. Vivaldi, on the other hand, requires no separate infrastructure.

Waldvogel and Rinaldi [22, 19] describe a spring relaxation technique to assign IDs to landmark nodes as part of the Mithos proximity-aware overlay network. A number of aspects of their algorithm require a centralized implementation.

Lighthouse [15] is an extension of GNP that is intended to be more scalable. Lighthouse, like GNP, has a special set of landmark nodes. Unlike GNP, a node that joins Lighthouse does not have to query those global landmarks. Instead, the new node can query any existing set of nodes to find its coordinates relative to that set, and then optionally transform those coordinates into coordinates relative to the global landmarks.

Priyantha *et al.* [16] describe a distributed node localization system for wireless sensor networks that uses spring relaxation. The sensors use ultrasound propagation times to measure inter-sensor distances and cooperate to derive coordinates consistent with those distances. Much of the algorithm is devoted to solving a problem that doesn't affect Vivaldi: the fact that two non-adjacent sensors cannot measure the distance between themselves makes it hard for the system to avoid letting the coordinate space double back on itself.

Rao *et. al.* [17] compute virtual coordinates for use in geographic forwarding in a wireless ad-hoc network. Their algorithm does not attempt to predict latencies; instead, the purpose is to make sure that directional routing works.

A number of peer-to-peer networks incorporate proximity routing [18, 3, 11]. We hope that many of these systems will benefit from using synthetic coordinates.

6 Conclusions and Discussion

Vivaldi is a simple, distributed algorithm for finding synthetic coordinates that accurately predict Internet latencies. Vivaldi is fully distributed; for example, it does not require a pre-selected subset of nodes to be designated as landmarks. Vivaldi is simple; it has only one tunable parameter. These properties make it easy to deploy Vivaldi in distributed systems: we have used Vivaldi to improve the performance of the Chord peer-to-peer lookup system.

Additional work remains to further understand and improve Vivaldi's performance. For example, we are exploring how nodes' communication patterns affect prediction accuracy. We are also interested in determining to what extent Internet latencies can be embedded in a N-dimensional Euclidean space and the reasons such an embedding is possible. Finally, we plan to modify additional distributed applications to take advantage of Vivaldi and quantify how much Vivaldi improves performance.

References

[1] Planetlab. www.planet-lab.org.

[2] BitTorrent. http://bitconjurer.org/BitTorrent/protocol.html.

[3] Miguel Castro, Peter Druschel, Y. C. Hu, and Antony Rowstron. Exploiting network proximity in peer-to-peer overlay networks. Technical Report MSR-TR-2002-82, Microsoft Research, June 2002.

[4] Russ Cox and Frank Dabek. Learning Euclidean coordinates for Internet hosts. www.pdos.lcs.mit.edu/˜rsc/6867.pdf, December 2002.

[5] Frank Dabek, M. Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Wide-area cooperative storage with CFS. In *Proc. 18th ACM Symposium on Operating Systems Principles (SOSP '01)*, October 2001. `http://www.pdos.lcs.mit.edu/chord`.

[6] P. Francis, S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, and L. Zhang. IDMaps: A global Internet host distance estimation service. *IEEE/ACM Transactions on Networking*, October 2001.

[7] Thomer Gil, Jinyang Li, Frans Kaashoek, and Robert Morris. Peer-to-peer simulator, 2003. Source available at: cvs.pdos.lcs.mit.edu.

[8] Krishna P. Gummadi, Stefan Saroiu, and Steven D. Gribble. King: Estimating latency between arbitrary Internet end hosts. In *Proc. of SIGCOMM IMW 2002*, November 2002.

[9] H. Hoppe. *Surface reconstruction from unorganized points*. PhD thesis, Department of Computer Science and Engineering, University of Washington, 1994.

[10] KaZaA media dekstop. `http://www.kazaa.com/`.

[11] John Kubiatowicz, David Bindel, Yan Chen, Steven Czerwinski, Patrick Eaton, Dennis Geels, Ramakrishna Gummadi, Sean Rhea, Hakim Weatherspoon, Westley Weimer, Chris Wells, and Ben Zhao. OceanStore: An architecture for global-scale persistent storage. In *Proceeedings of the Ninth international Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2000)*, pages 190–201, Boston, MA, November 2000.

[12] P. Mockapetris and K. J. Dunlap. Development of the Domain Name System. In *Proc. ACM SIGCOMM*, pages 123–133, Stanford, CA, 1988.

[13] Eugene Ng. Gnp software, 2003. Source available at: `http://www-2.cs.cmu.edu/~eugeneng/research/gnp/software.html`.

[14] T. S. Eugene Ng and Hui Zhang. Predicting Internet network distance with coordinates-based approaches. In *Proceedings of IEEE Infocom 2002*, 2002.

[15] Marcelo Pias, Jon Crowcroft, Steve Wilbur, Tim Harris, and Salem Bhatti. Lighthouses for scalable distributed location. In *IPTPS*, 2003.

[16] N. Priyantha, H. Balakrishnan, E. Demaine, and S. Teller. Anchor-free distributed localization in sensor networks. Technical Report TR-892, MIT LCS, April 2003.

[17] Ananth Rao, Sylvia Ratnasamy, Christos Papadimitriou, Scott Shenker, and Ion Stoica. Geographic routing without location information. In *ACM MobiCom Conference*, September 2003.

[18] Sylvia Ratnasamy, Mark Handley, Richard Karp, and Scott Shenker. Topologically-aware overlay construction and server selection. In *Proceedings of IEEE Infocom 2002*, 2002.

[19] Roberto Rinaldi and Marcel Waldvogel. Routing and data location in overlay peer-to-peer networks. Research Report RZ–3433, IBM, July 2002.

[20] Yuval Shavitt and Tomer Tankel. Big-bang simulation for embedding network distances in Euclidean space. In *Proc. of IEEE Infocom*, April 2003.

[21] Ion Stoica, Robert Morris, David Liben-Nowell, David Karger, M. Frans Kaashoek, Frank Dabek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Transactions on Networking*, pages 149–160, 2002.

[22] Marcel Waldvogel and Roberto Rinaldi. Effi cient topology-aware overlay network. In *Hotnets-I*, 2002.

[23] Limin Wang, Vivek Pai, and Larry Peterson. The Effectiveness of Request Redirecion on CDN Robustness. In *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation*, Boston, MA USA, December 2002.